Manual técnico:

Clases:

Lamadas:

```
class Llamadas{
    constructor(id_operador, nombre_operador, estrellas, id_cliente, nombre_cliente){
        this.id_operador = id_operador;
        this.nombre_operador = nombre_operador;
        this.estrellas = estrellas;
        this.id_cliente = id_cliente;
        this.nombre_cliente = nombre_cliente;
    }
}

export default Llamadas;
```

en esta clase se guardarán todos los datos del archivo ingresado para lectura, y se guardaran en una lista declarada en el index llamada registroLlamadas

Operador:

```
class Operadores{
    constructor(id_operador, nombre_operador, rendimiento){
        this.id_operador = id_operador;
        this.nombre_operador = nombre_operador;
        this.rendimiento = rendimiento;
    }
}

export default Operadores;
```

De la lista registroLlamdas se sacaron lo datos de los operadores y se guardaran en una lista declarada en el index llamada registroOperadores

Funciones:

Generador:

```
const generadorReporte = (nombreArchivo, informacion) => {
    let htmlCode = `<!DOCTYPE html>
    <html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>Reporte</title>
    </head>
    <body>      You, 8 hours ago • se modifico la funcion generadorReporte  para
        <table border="1">
            <tr>
                <th colspan="2">Cliente</th>
                <th colspan="3">Operador</th>
            </tr>
            <tr>
                <th>Id</th>
                <th>Nombre</th>
                <th>Id</th>
                <th>Nombre</th>
                <th>Calificación</th>
            </tr>`
```

Esta función recibirá de parámetro un nombre en este caso reporte historial de llamadas y la lista registroLlamadas, esta función tendrá declarado al inicio una tabla en html seguido de un forEach que recorrerá la lista registroLlamada buscado los datos del cliente (id, nombre) y del operador (id, nombre, estrellas) y concatenándola a la tabla declarada al inicio

```
    informacion.forEach(registro => {
        htmlCode += `<tr>
            <td>${registro.id_cliente}</td>
            <td>${registro.nombre_cliente}</td>
            <td>${registro.id_operador}</td>
            <td>${registro.nombre_operador}</td>
            <td>${registro.estrellas}</td>
        </tr>
        `
    })
```

Por ultimo el cierre de la tabla y un stream para generar un documento con la librería fs en la carpeta Salidas en formato HTML

```
    htmlCode += `</table>
        </body>
        </html>`

    try{
        const stream = fs.createWriteStream(`./Salidas/${nombreArchivo}.html`, 'utf-8');
        stream.write(htmlCode);
        stream.end();
    }catch (error){
        console.log("Error al escrbir el archivo")
    }
}
```

generadorClientes:

esta función recibe los mismos parámetros que la función Generador, la única diferencia es el forEach que es este caso solo busca el id y nombre del cliente en la lista registroLlamadas

```
import fs from 'fs';

const generadorClientes = (nombreArchivo, informacion) => {
    let htmlCode = `<!DOCTYPE html>
    <html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>Reporte</title>
    </head>
    <body>
        <table border="1">
            <tr>
                <th colspan="2">Cliente</th>
            </tr>
            <tr>
                <th>Id</th>
                <th>Nombre</th>
            </tr>`

    informacion.forEach(registro => {
        htmlCode += `<tr>
            <td>${registro.id_cliente}</td>
            <td>${registro.nombre_cliente}</td>      You, 6 hours ago • Se creo la funcion g
        </tr>
        `
    })

    htmlCode += `</table>
        </body>
        </html>`

    try{
        const stream = fs.createWriteStream(`./Salidas/${nombreArchivo}.html`, 'utf-8');
        stream.write(htmlCode);
        stream.end();
    }catch (error){
        console.log("Error al escrbir el archivo")
    }
}

export default generadorClientes;
```

generadorOperadores:

esta función recibe los mismos parámetros que la función Generador, la única diferencia es el forEach que es este caso solo busca el id y nombre de los operadores en la lista registroLlamadas

```
import fs from 'fs';

const generadorOperadores = (nombreArchivo, informacion) => {
    let htmlCode = `<!DOCTYPE html>
    <html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>Reporte</title>
    </head>
    <body>
        <table border="1">
            <tr>
                <th colspan="2">Operador</th>
            </tr>
            <tr>
                <th>Id</th>
                <th>Nombre</th>
            </tr>`

    informacion.forEach(registro => {
        htmlCode += `<tr>
            <td>${registro.id_operador}</td>
            <td>${registro.nombre_operador}</td>      You, 6 hours ago • se creo la funcio
        </tr>
        `
    })

    htmlCode += `</table>
        </body>
        </html>`

    try{
        const stream = fs.createWriteStream(`./Salidas/${nombreArchivo}.html`, 'utf-8');
        stream.write(htmlCode);
        stream.end();
    }catch (error){
        console.log("Error al escrbir el archivo")
    }
}
```

generadorRendimiento:

esta función recibe los mismos parámetros que la función Generador, la única diferencia es el forEach que es este caso solo busca el id y nombre de los operadores en la lista registroOperadores

```
const generadorRendimiento = (nombreArchivo, informacion) => {      You, 5 hours ago • s
    let htmlCode = `<!DOCTYPE html>
    <html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>Reporte</title>
    </head>
    <body>
        <table border="1">
            <tr>
                <th colspan="3">Operador</th>
            </tr>
            <tr>
                <th>Id</th>
                <th>Nombre</th>
                <th>Rendimiento</th>
            </tr>`

    informacion.forEach(registro => {
        htmlCode += `<tr>
            <td>${registro.id_operador}</td>
            <td>${registro.nombre_operador}</td>
            <td>${registro.rendimiento}</td>
        </tr>
        `
    })

    htmlCode += `</table>
        </body>
        </html>`

    try{
        const stream = fs.createWriteStream(`./Salidas/${nombreArchivo}.html`, 'utf-8');
        stream.write(htmlCode);
        stream.end();
    }catch (error){
        console.log("Error al escrbir el archivo")
    }
}
```

registroLlamadas:

```
const contenido = fs.readFileSync(`./Entradas/${archivo}`, 'utf-8');

let lineas = contenido.split(/\r?\n/)
lineas = lineas.slice(1);       You, 4 days ago • se imprime el archivo entrada.csv en un arreglo…

for(let i = 0; i < lineas.length; i++){
    lineas[i] = lineas[i].split(/,/);
    lineas[i][2] = lineas[i][2].split(/;/);
    lineas[i][2] = lineas[i][2].filter(e => e === 'x').length;
    registroLlamadas.push( new Llamadas(lineas[i][0], lineas[i][1], lineas[i][2], lineas[i][3], lineas[i][4]));
}
```

Declaramos contenido y con la librería fs hacemos que lea el archivo en la ruta de texto marcada y que se almacene como una cadena de texto, después declaramos líneas que es un array donde cada elemento es una línea del archivo usando Split para dividirlo el contenido usando la expresión regular \r\n, luego iniciamos un bucle en líneas que divide la línea actual por comas “,” después accedemos al tercer elemento del array y lo dividimos por cada punto y com “;” usamos filter para filtrar los elementos del ese array que coincidan con “x” y por ultimo creamos una nueva instancia Llamadas y la agregamos a la lista registroLlamadas

registroOperadores:

```
let datosOperadores = registroLlamadas.reduce(function(acc, llamada) {
    let nombre = llamada.nombre_operador;
    if (!acc[nombre]){
        acc[nombre] ={
            id_operador: llamada.id_operador,
            llamadasAtendidas: 0
        };
    }
    acc[nombre].llamadasAtendidas += 1;
    return acc;
}, {});

for (let nombre in datosOperadores){
    const data = datosOperadores[nombre];
    const rendimiento = (data.llamadasAtendidas / registroLlamadas.length) * 100;

    registroOperadores.push( new Operadores(data.id_operador, nombre, parseFloat(rendimiento.toFixed(2))));
}
```

Usamos reduce() para procesar cada elemento y acumular el reslutado del array anterior registroLlamadas, el acc es el acumulador que guardara el resultado parcial y llamadas es el elemento actual del array que esta siendo procesado, el ¡acc[nombre] verifica si el operador ya existe en el acumulador

Si el operador no existe en el acumulador, crea una nueva entrada y aumenta el contador de llamadas atendidas para este operador esto se ejecuta para cada llamada en el array, el return acc devuelve el acumulador actualizado para la siguiente iteración

El for itera sobre todas las propiedades del objeto, obtiene los datos del operador actual y calcula el rendimiento el .toFixed(2) limita el porcentaje a 2 decimales, parseFloat() convierte la cadena a un numero new operador crea una instancia del objeto operador y push agrega el nuevo objeto al array registroOperador